Target Libraries

Target CPU Boards

User's Manual

Contents

Revision History

Manual	Version	Comments	
Issue A	V 1 Iss. 1	961105	First released in this format
Issue B	V 2 Iss. 1	-	Chip select, vArCS(), and default bus vArSteBus(), functions added. (This issue not released at Arcom Ltd.)
Issue C	V 3 Iss. 1	970730	ECO2624 Corrected wArOpen() description, wArTmrInit params, Added vArWatchDogInit(), timer examples. (Arcom Inc. Revision date: May 16, 1997.)

Introduction

The Arcom Library

Arcom Control Systems produces a complete range of high quality Target Processors In order to support these processor cards as fully, a complimentary set of high quality software support products is also available. This software library forms part of this software support alongside other drivers in the range as a self-contained set of utilities which utilises the complete hardware capabilities.

With the vast number of potential hardware and software platforms common in the Industrial PC market place this software support has been developed to support the most common compilers available - Microsoft Visual C and Borland C, running on 386 CPUs to Pentium CPUs, in both DOS and Windows operating systems.

Although it is expected that the user of this software has a working knowledge of their PC and compiler, this library and manual has been written so as to allow a user new to PC I/O programming to quickly achieve their goals. More advanced users will be able to refer to the function library section directly to gain all the information they require.

This software is presented to the user in the form of standard static library files. This library can then be incorporated into custom application code giving the user all the software facilities needed to use all the hardware features included on an Arcom PCbus I/O card. Many of the boards have a built in flexibility that allow the user to achieve unique technical solutions. To avoid constricting the user to a limited selection of features the library gives the user low-level control of the board for direct programming control.

Once the board has been initialised successfully the library uses a powerful yet simple access system using a board identification number or 'handle'. This handle is then used in order to allow a general purpose standard function call to perform functions specific to the chosen card.

Highlights

This library provides the following features :-

- Board Initialisation
- Input/Output enable after reset
- User LED control
- Timer control
- Interrupt driven serial I/O
- Comprehensive support for Counter/Timers
- Watchdog function support for the PCSYSCON
- Complete function listing
- Command Line support for Borland C compilers

How to Use This Manual

Who is it for?

This manual is designed to provide an easy to use reference for new, intermediate, and experienced users. In order to achieve this the manual has been divided into sections. For a new user this manual should be read through in section order, taking the step-by-step approach outlined in the installation and getting started sections.

For the more advanced user the fast start procedure and functional description sections may be of greatest use, and users wishing to take greater control over their own software can make use of the advanced users information later in the manual. This information will help with using the more specialised features of the board and lower level access functions for specific needs not covered in the library.

The major portion of this manual is contained in Section 5, the function library list itself. The library list itself is self contained and in alphabetical order, the surrounding sections guide the user in the ways to get the most from the large selection available.

The 'Getting Started' section is split into two successive parts, referring firstly to the Borland C specific part and then secondly the Microsoft C part. This allows you to follow only the relevant sections and skip those not required for your own choice of compiler.

Which Compilers?

The library supports the use of 'command-line' versions of the above compilers. This allows the greatest number of possible application platforms to be supported and ensures the user is presented with the most easily customised and powerful options. If the user is familiar with a built in editor and compiler environment all the attached libraries and programming examples can be used without the need for any modifications to the libraries themselves. This means the user will only need to know how to perform the software links to the library as supplied as standard.

Some users may be unfamiliar with a command-line compiler. All compilers supported by this library have command-line versions shipped and installed as part of the complete package, however new users are encouraged by the compiler vendors to make use of an integrated development environment rather than use the command line version. Experienced users are more likely to prefer to use the command-line version, together with their favourite source editor, it is in fact a very straightforward process that is clearly described in the getting started section later on in this manual.

Scope of Supported Boards

The list below shows the boards supported within the scope of this library. All commonly used boards are listed, however those that fall outside this list are not supported within this software library. If you have a different board than the ones listed below please refer to your hardware manual and contact Arcom for software advice.

BOARD	FUNCTION
• Target386	Intel 386Ex based Target board
• Target188	Intel 188EB base Target board

Typefaces used in this manual

The following typefaces are used throughout this manual:-

TYPEFACE	USAGE
• Normal Typeface	used for the main body of text
• `Program Statements`	`used for source code text`
• `Command Line Inputs`	`used when text is to be inputted at the command line`
• *File Names*	*Used when referring to file names*

General users information

Use the following short guide to this software library when you begin programming. This guides you through the general use of all supported Target Processors.

Base Address

Make sure your base address settings are unique within your I/O map space, and do not conflict with locations already in use within your Target Environment. Make sure multiple I/O boards have different settings and that your software knows how they are set. It is important to make a note of your settings as you will not be able to refer to the boards when your PC's cover is replaced.

Links

Make sure your link settings are made correctly for the mode in which you need to use the card. Apart from the base address settings as above, functions that particularly need these to be correct are digital I/O direction and reset settings, analogue channel scaling, interrupt generation and counter timer set-ups.

Initialisation

The first task your software must perform is board initialisation. This sets up the board and the software library functions to a known state, and no I/O functions will work without the board being initialised.

Conventions

C Language Notation

Throughout this document, reference will be made to 'C' data declarations and structures. Unless otherwise stated, the following definitions of data types will be assumed:

- Byte ordering within data entities greater than 8 bits will be of Little Endian format type.
- 'char' data types are 8 bit unsigned data entities.
- 'unsigned int' and 'int' data types are 16 bit entities.
- 'long' data types are 32 bit unsigned entities.
- 'float' data types are 32 bit IEEE 754 format entities.

Data Types

The following definitions are used to further standardise the data types used:

System Data Types				Description
typedef	signed	char	**SCHAR;**	// 8 Bit signed character
typedef	signed	char	**SINT8;**	// 8 Bit signed integer
typedef	signed	char	**SBYTE**	// 8 Bit signed hex value
typedef	unsigned	char	**UCHAR**	// 8 Bit unsigned character
typedef	unsigned	char	**UINT8**	// 8 Bit unsigned integer
typedef	unsigned	char	**UBYTE**	// 8 Bit unsigned hex value
typedef	unsigned	char	**FLAG**	// 8 Bit unsigned hex value
typedef	unsigned	char	**BOOL**	// 8 Bit unsigned hex value
typedef	signed	short	**SINT16**	// 16 Bit Signed integer
typedef	unsigned	short	**UINT16**	// 16 Bit Unsigned integer
typedef	signed	long	**SINT32**	// 32 Bit signed integer
typedef	unsigned	long	**UINT32**	// 32 Bit unsigned integer
typedef	signed	float	**REAL32**	// 32 Bit floating point value
typedef	signed	double	**REAL64**	// 64 Bit floating point value

The system data types SCHAR, SINT8, and SBYTE are treated the same and only exist to provide further documentation as to their contents. This also applies to the data types UCHAR, UINT8, and UBYTE.

Variable Naming

The following prefixes will be used to clarify the type of variables:

Variable Names		
Prefix	**Example**	**Description**
s	`UCHAR sTelnetLogin[]="user";`	`// Null Terminated string`
b	`UBYTE bSlipParity = 0;`	`// 8 Bit byte value`
f	`FLAG fExit;`	`// (TRUE/FALSE) flag`
w	`SINT16 wUintAdr = 54;`	`// 16 bit word value`
u	`UINT16 uChan;`	`// 16 bit word value`
l	`SINT32 1Baud = 384001;`	`// 32 bit long word value`
ul	`UINT32 ulMsTime;`	`// 32 bit long word value`
r	`REAL32 rHartValue = 38.4;`	`// 32 bit Float value`
d	`REAL64 dHartValue = 38.4;`	`// 64 bit Float value`
a	`UINT16 aWidgets[2];`	`// Array of entities`
i	`UINT16 iWidgets = 0;`	`// Index into array`
c	`UINT16 cWidgets=sizeof (aWidgets);`	`// Count of bytes`
p	`UINT16 *pWidgets=aWidgets;`	`// Data Pointer`
fp	`UINT16 far *fpWidgets = (far)*aWidgets;`	`// Far Data Pointer`
v	`void vArTargetInit();`	`// void declaration`

Variable names use one of the above type prefixes with each "word" capitalised. Only the first letter of an Acronym is capitalised.

How to contact Arcom

If you need to speak to Arcom about any matter that arises from the usage of this software or its associated hardware you may contact our technical support department in the following ways:

Before you contact us!

Please have the following information close to hand to help us assist you efficiently:-

- Make, version and issue numbers of your compiler and operating system
- General description of your hardware set-up (especially CPU and clock speed).
- Version and issue numbers of your software library and all your Arcom hardware.
- A description of the problem.

You can use one of the following systems to contact us:-

By telephone

Call our customer support hotline on UK 01223 412 428
For international callers the number is +44 1223 412 428

By fax

Our customer support fax number is 01223 403 400

By e-mail

Our customer support e-mail address is support@arcom.co.uk

Bulletin Board

You can also use our BBS service by calling 01223 415 146

Busy Times

During busy times it helps us process enquiries more effectively if you can write down as much information as you can and send it by fax. Alternatively you may prefer to make use of our Voice Mail system which will take a message if all lines are busy.

Board Support Functions

vArTargetInit()

Set up a known working environment.

Synopsis
```
#include ["Ar386Ex.h" | "Ar188Eb.h"]
void vArTargetInit ( void );
```

Arguments none

Function This function is used to set up a known working environment for use with a particular target.

386EX The following operating parameters are set:
All interrupts in the Master Interrupt controller are masked
All interrupts in the Slave Interrupt controller are masked.
The spurious interrupt vector is set to a stub function.

188EB The following operating parameters are set:
1) Sets Port #2 usage as:
P2:7 P2:6 = Port Function
P2:3 Latch to Pin
2) Set P2 direction to operate:
RED/GREEN LED P2:6 & P2:7 Outputs
WatchDog P2:3 Output
User Input Links P2:4 & P2:5 Inputs
3) Set-up CS0 & CS1 for use with the onboard SCC chip. GCS0 is used for control/data access and will begin @ 0xf880 for SCIM88 compatibility and will extend to 0xf8c0. GCS1 is used for Int Ack of the SCC and will be active @ 0xf900 to 0xf9c0. The address of 0xf900 again is for SCIM88 compatibility.
4) Set-up CS3 for use with the onboard Parallel I/O.
0xfa00 - Write Only output port.
0xfa01 - Read Only input port.
0xfa02 - Read/Write I/O port.
5) Set up an STE I/O access area starting @ 0x0300 and ending @ 0x03c0. Using GCS4 for this purpose.
NOTE!!! when using the STEbus, the external ready (RDY) bit needs to be enabled.

Returns No values are returned.

Example See example program *demo[386|188].c*

See Also

vArWatchDogInit()

Initialise Processor watchdog.

Compatibility 386EX, 188EB

Synopsis

```
#include "Ar386ex.h"
or
#include “Ar188eb.h”

void vArWatchDogInit( UINT32 ulTimer );
```

Arguments

UINT32 ulTimer
Timeout constant in processor clock cycles.

Function

Initialises the watchdog timeout constant but does not start the countdown.
The watchdog timer will start counting down from the first call to vArWatchDog().
If the watchdog is already running, the timeout constant will not be changed.

NB. Target188EB watchdog timeout is board specific at 1.2s and will not be altered by calling this function.

Target386EX watchdog timeout count may be between 8 counts and 0xFFFFFFFF counts (242ns to 130.2s on a 33Mhz system). To find the watchdog time constant use:

Timeout Delay (s) = Timeout Constant / Processor Clock Frequency

Returns none

Example See example program *demo[386].c*

See Also

vArWatchDog()

Reset Processor watchdog.

Compatibility 386EX, 188EB

Synopsis

```
#include "Ar386ex.h"
or
#include "Ar188eb.h"

void vArWatchDog( void );
```

Arguments none

Function Performs the processor watchdog 'lockout sequence' which reloads the watchdog and keeps it from resetting the CPU.

If the Target 386EX watchdog has not been initialised prior to calling this function the time out delay will be set to 230 processor cycles » 32.5s.

The Target188EB watchdog is hardware selectable using link LK14. This routine must be called repeatedly to strobe the watchdog at intervals of less than 1.2s, in order to ensure that a watchdog system reset is not generated.

Returns none

Example See example program *demo[386].c*

See Also

vArReboot()

Reset processor.

Compatibility 386EX, 188EB

Synopsis

```
#include "Ar386ex.h"
or
#include “Ar188eb.h”

void vArReboot( void );
```

Arguments none

Function Initialises the watchdog then disables all interrupts to force the watchdog to reset the CPU. (A system reset will not occur if link settings are such that the watchdog is disabled)

Returns none

Example See example program *demo[386|188].c*

See Also

vArLed()

Ignite or extinguish the on-board LEDs.

Compatibility 386EX, 188EB

Synopsis

```
#include "Ar386ex.h"
or
#include “Ar188eb.h”
```

void vArLed(UINT16 uLed, FLAG fState);

Arguments

UINT16 uLed
LED to manipulate. Use LED_GREEN or LED_RED predefined constants.

FLAG fState
State of LED. TRUE for igniting LED. FALSE for extinguishing LED.

Function Manipulates one of the two (LED_RED or LED_GREEN) on-board LEDs to a TRUE (on) or FALSE (off) state.

Returns none

Example See example program *demo[386|188].c*

See Also

fArUsrLnk()

Read status of User Links.

Compatibility 386EX, 188EB

Synopsis

```
#include "Ar386ex.h"
or
#include “Ar188eb.h”

FLAG fArUsrLnk( UINT16 uUsrLnk );
```

Arguments

UINT16 uUsrLnk
User Link to read. Use USRLNK1 or USRLNK2 predefined constants.

Function Read back status of supplied user-link.

Returns Zero if Link is omitted.
Non-zero if Link is installed.

Example See example program *demo[386|188].c*

See Also

vArSteBus()

Set the default bus to the STE bus or PC/104 bus.

Compatibility `386EX non SBC variant.`

Synopsis

```
#include "Ar386ex.h"
```

void vArSteBus (FLAG fSteBus)

Arguments **FLAG fSteBus**
TRUE (1) = The STE bus is the default.
FALSE (0) = The PC/104 bus is the default.

Function Allows user to set the STE bus or the PC/104 bus as the default bus.

Returns None.

Example See vArCS() for 386EX, Example 2

See Also vArCS

vArCS()

Map memory or I/O blocks.

Compatibility 188EB

Synopsis

```
#include "Ar188eb.h"
```

void vArCS (UINT8 bCs, FLAG fEnable, FLAG fIStop, FLAG fMem, FLAG fRdy, UINT8 bWS, UINT32 ulStrtAddr, UINT32 ulStopAddr)

Arguments

UINT8 bCs;
Assign a chip select to this block. (Valid range: 0-9)
FLAG fEnable;
TRUE (1) = Enable this chip select.
FALSE (0) = Disable this chip select.
FLAG fIStop;
TRUE (1) = Ignore upper address limits and allow maximum access.
FALSE (0) = Stop at upper address limit.
FLAG fMem;
TRUE (1) = The area specified is a memory block.
FALSE (0) = The area specified is an I/O block.
FLAG fRdy
TRUE (1) = Set external ready signal for STEbus access.
FALSE (0) = Don't need external ready signal for non STEbus access.
UINT8 bWS
Number of wait states (Valid range: 0 to 15).
UINT32 ulStrtAddr;
32 bit starting address of block.
UINT32 ulStopAddr;
32 bit upper address limit of block. (See Notes 1 & 2 below)

Notes:

1 Memory blocks can be specified in 1K byte increments up to 1Meg. in size.

2 I/O blocks can be specified in 64 byte increments up to 64Kb. in size.

Function Allows user to flexibly map 10 independent memory and I/O blocks on the STE bus.

Returns None.

Example See examples on next page and the program *demo188.c*

See Also ----

An example of mapping memory on the 188EB target.

void vArCS (UINT8 bCs, FLAG fEnable, FLAG fIStop, FLAG fMem, FLAG fRdy, UINT8 bWS, UINT32 ulStrtAddr, UINT32 ulStopAddr)

Example : Memory on the STE bus.

```
//
// Map a block of STEbus memory from 50000h to 58000h.
//
#include <ArTypes.h>
#include <Ar188eb.h>

UINT8  bCs = 4;                          // Pick Chip select #4 for this block.
FLAG   fEnable = TRUE;                   // Enable this CS block.
FLAG   fIStop = FALSE;                   // Stop at upper address limit.
FLAG   fMem = TRUE;                      // This is a memory block.
FLAG   fRdy = TRUE;                      // Set Rdy flag for STEbus access.
UINT8  bWS = 1;                          // Set 1 wait state but Rdy will rule.
UINT32 ulStrtAddr = 0x50000L; // Starting address of block.
UINT32 ulStopAddr = 0x58000L; // Upper limit address(+1) of block.

// Now call the chip select function.

vArCS (bCs, fEnable, fIStop, fMem, fRdy, bWS, ulStrtAddr, ulStopAddr)

// Memory at 50000h for 32K is now available for access on the STE bus.
```

vArCS()

Map memory or I/O blocks to STE bus or PC/104 bus.

Compatibility 386EX

Synopsis

```
#include "Ar386ex.h"
```

void vArCS (UINT8 bCs, FLAG fEnable, FLAG fSte, FLAG fMem, UINT32 ulAddr, UINT32 ulSize)

Arguments

UINT8 bCs;
Chip select to manipulate. (Valid range: 0-7)
FLAG fEnable;
TRUE (1) = Enable this chip select.
FALSE (0) = Disable this chip select.
FLAG fSte;
TRUE (1) = The block is for the STE bus.
FALSE (0) = The block is for the PC/104 bus.
FLAG fMem;
TRUE (1) = The area specified is a memory block. (See Note 1 below)
FALSE (0) = The area specified is an I/O block. (See Note 2 below)
UINT32 ulAddr;
32 bit starting address of I/O or memory block. (See Notes below)
UINT32 ulSize;
32 bit size of I/O or memory block. (See Notes below)
Notes:

1 Memory starting address must be on a (2)**n Kbytes memory address boundary and size must be specified in (2)**n Kbytes. (Where n = 1 to 15.)

2 I/O starting address must be on a (2)**n byte I/O address boundary and size must be specified in (2)**n bytes. (Where n = 1 to 15.)

Function Allows user to map 8 memory and I/O blocks on-board or on the PC/104 bus or the STE bus.

Returns None.

Example See examples on next page and the program *demo386.c*

See Also vArSteBus

Two examples of mapping memory and I/O blocks on the 386EX target.

```
void vArCS(UINT8 bCs, FLAG fEnable, FLAG fSte, FLAG fMem,
           UINT32 ulAddr, UINT32 ulSize)
```

Example #1: Memory on the STE bus.

```
//
// Map a 32Kb block of memory on the STE bus at 50000h.
//
#include <ArTypes.h>
#include <Ar386Ex.h>

UINT8  bCs = 3;              // Chip select #3 is available for this block.
FLAG   fEnable = TRUE;       // Enable this CS block.
FLAG   fSte = TRUE;               // This is an STE bus block
FLAG   fMem = TRUE;               // This is a memory block
UINT32 ulAddr = 0x50000L;         // Start at address 50000h.
UINT32 ulSize =  0x08000L;   // Block size = 32k bytes.
// Now call the chip select function.

vArCS(bCs, fEnable, fSte, fMem, ulAddr, ulSize);
// Memory at 50000h for 32K is now available for access on the STE bus.
```

Example #2: I/O on the PC/104 bus.

```
//
// Map a 2 byte I/O block at 100h on the PC/104 bus.
//
/#include <ArTypes.h>
#include <Ar386Ex.h>

//  First make the PC/104 bus the “NON-default” bus.
vArSteBus (TRUE);            // Make the STE bus the default bus.

// Now map a 2 byte I/O block at 100h on the PC/104 bus.
UINT8 bCs = 4;                    // Chip select #4 is available for this
block.
FLAG fEnable = TRUE;         // Enable this block to be active.
FLAG fSte = FALSE;           // This is a PC/104 block.
FLAG fMem = FALSE;           // This is an I/O block.
UINT32 ulAddr = 0x100;  // Start at 100h.
UINT32 ulSize = 2;           // Block size is 2 bytes.
// Now call the chip select function.

 vArCS(bCs, fEnable, fSte, fMem, ulAddr, ulSize);
// I/O address 100h-101h is now available for access on the PC/104 bus.
```

wArTmrInit()

Initialise timers for target boards.

Compatibility 386EX and 188EB

Synopsis #include ["Ar386ex.h" | "Ar188eb"]

SINT16 wArTmrInit (UINT16 timer, UINT16 mode, UINT16 time, UINT16 time2,
void interrupt(__far *vpTmr)(void))

Arguments

UINT16 timer;
Number of timer to configure. (Valid range: 0-2)
UINT16 mode;
Operating mode of timer. (ONESHOT or ASTABLE)

UINT16 Time;
Count value for the timer

UINT16 Time2;
Count value for B phase of timer. (Always 0 for 386EX)

***vpTmr;**
Pointer to interrupt routine for this timer.

Description Initialises a timer to run in a given mode but does not actually start the the timer.
Notes:
If a null pointer is passed for the interrupt routine, the interrupts will be disabled.
An interrupt will be generated upon terminal count.

Clock frequency for Target188EB CLKIN=**25Mhz**
Clock frequency for Target386EX CLKIN=**33Mhz**

For the 188EB:
If time2 is not a "0", timer will be run in duty cycle mode.

Returns
SUCCESS - Timer has been set up as specified.
TMR_NUMBER - Timer number passed is not a 0,1 or 2.
TMR_MODE - Mode passed is not ONESHOT or ASTABLE
TMR_COUNT - Count passed must not be a "1" in ASTABLE mode.

Example See example #3 at the end of this section for formulation of count values
See examples in the programs *demo386.c and Demo188.c*

wArTmrPrescale()

Set prescale count for timer

Compatibility 386EX and 188EB

Synopsis #include ["Ar386ex.h" | "Ar188eb"]

SINT16 wArTmrPrescale (UINT16 time, FLAG timer0, FLAG timer1);

Arguments

UINT16 time;
Count value for the prescaler.
This is the ACTUAL divisor for the internal clock
Valid ranges are:
386EX range is 5 to 513 and 188EB range is 0 to 65535
FLAG timer0; (Used for the 188EB only.)
Set if timer0 is to be prescaled.

FLAG timer1; (Used for the 188EB only.)
Set if timer1 is to be prescaled.

Description Initialises a prescaler and starts it running. If this feature is used in the 188EB, Timer2 will automatically be the prescaler thus precluding it from being used as a timer.
NB This function MUST be called to initialise prescaling before timers are started.
The minimum prescale divisor on the Target386ex (33Mhz) is 5 as the timer/counter unit will not operate at a frequency in excess of 8Mhz

Returns

SUCCESS - Prescaler has been set up as specified.
TMR_START - Prescaler timer is already running. (for 188EB only)
TMR_COUNT - Invalid count passed. (for 386EX only)

Example See example #3 at the end of this section for formulation of prescale values
See examples in the programs *demo386.c and Demo188.c*

wArTmrStart()

Start the selected timer.

Compatibility `386EX and 188EB`

Synopsis

```
#include ["Ar386ex.h" | “Ar188eb”]

SINT16 wArTmrStart  (UINT16  timer);
```

Arguments

UINT16 timer;
The number of the timer to start. (0, 1 or 2)

Description This function starts the selected timer. The timer must have been already setup with the function wArTmrInt().

Returns

SUCCESS - Timer has been started.
TMR_NUMBER - Timer number passed was not a 0, 1 or 2.
TMR_START - Timer is already running.
TMR_NO_INIT - Selected timer is not initialised.

Example See examples in the programs *demo386.c and Demo188.c*

wArTmrStop()

Stop the selected timer.

Compatibility 386EX and 188EB

Synopsis

```
#include ["Ar386ex.h" | “Ar188eb”]
```

SINT16 wArTmrStop (UINT16 timer);

Arguments

UINT16 timer;
The number of the timer to stop. (0, 1 or 2)

Description This function stops the selected timer. The timer must be running.

Returns

SUCCESS - Timer has been stopped.
TMR_NUMBER - Timer number passed was not a 0, 1 or 2.
TMR_STOP - Timer was not running.

Example See examples in the programs *demo386.c and Demo188.c*

Example #3: Formulation of Prescale/Count Values (386EX)

The formula to use when defining interrupt periods is shown below:

$$\text{Interrupt Period} = \left(\frac{\text{[Prescaling Value] x [Terminal Count Value]}}{\text{Internal Clock Frequency}} \right)$$

NB. The internal clock frequency for a Target386EX is fixed at 33Mhz.
The internal clock frequency for a Target188EB is fixed at 25Mhz.

Eg. To generate a series of interrupts at 100µs intervals

(100µs) * (33 Mhz) = (Prescaling Value) * (Terminal Count Value)

It is necessary to find values for the above, match the LHS.

As the prescaling value may not be less than 5 on the Target386EX, this is a valid starting point.

(Terminal Count Value) = (100µs) * (33 Mhz) / (5) = 660 (Must be an Integer)

Having found the required values for prescaling and terminal count we can now insert them into our code:

```
//
// Prescale timers.
//

wArTmrPrescale(5, FALSE, FALSE);  // OUR PRESCALING VALUE

//
// Initialise timer 0.
//

wArTmrInit ( 0,  // valid timers 0-2
             ASTABLE,    // modes are ONESHOT or ASTABLE
             660,         // OUR TERMINAL COUNT VALUE
      // dummy parameter for compatibility with 188eb
             (void far interrupt(*)(void))vTimer0Isr);

//
// Start timer 0.
//

wArTmrStart (0);

//
// Rest of code ...
//


//
// Stop timer 0.
//

wArTmrStop (0);
```

```
//
// Example of skeleton ISR
//

void interrupt vTimer0Isr (void)
{
  disable();                    // Interrupts disabled

  // Handle interrupt occurrences here ..

  vArEoi(TMR0_IRQ);             // Perform End-of-Interrupt

  enable();                    // Interrupts enabled
}
```

Example #4: Formulation of Prescale/Count Values (188EB)

The formula to use when defining interrupt periods is shown below:

$$\text{Interrupt Period} = \left(4x \ \frac{[\text{Prescaling Value}] \times [\text{Terminal Count Value}]}{\text{Internal Clock Frequency}} \right)$$

NB. The internal clock frequency for a Target386EX is fixed at 33Mhz.
The internal clock frequency for a Target188EB is fixed at 25Mhz.

Eg. To generate a series of interrupts at 100µs intervals

(100µs) * (25 Mhz) / 4= (Prescaling Value) * (Terminal Count Value)

It is necessary to find values for the above, match the LHS.

We do not need to prescale using timer 2 - However to illustrate let us choose to prescale by a factor of 5.

(Terminal Count Value) = [(100µs) * (25 Mhz) / (4)] / 5= 125 (Must be an Integer)

Having found the required values for prescaling and terminal count we can now insert them into our code:

```
//
// Prescale timers.
//

wArTmrPrescale(5, TRUE, FALSE);  // OUR PRESCALING VALUE
                                 // SET TO PRESCALE ONLY TIMER 0

//
// Initialise timer 0.
//

wArTmrInit ( 0,  // valid timers 0-2
             ASTABLE,    // modes are ONESHOT or ASTABLE
             125,         // OUR TERMINAL COUNT VALUE
             0,           // no duty cycle
             (void far interrupt(*)(void))vTimer0Isr);

//
// Start timer 0.
//

wArTmrStart (0);

//
// Rest of code ...
//


//
// Stop timer 0.
//

wArTmrStop (0);
```

```
//
// Example of skeleton ISR
//

void interrupt vTimer0Isr (void)
{
  disable();                    // Interrupts disabled

  // Handle interrupt occurrences here ..

  vArEoi(TMR0_IRQ);             // Perform End-of-Interrupt

  enable();                    // Interrupts enabled
}
```

Flash Support Functions for 386EX

fFlashInit()

Initialise the Flash device for operation.

Compatibility 386EX

Synopsis `#include "Ar386Ex.h" and "ArFlash.h"`

FLAG fFlashInit (void);

Arguments **None**

Description Set up the hardware specific registers for the operation of flash memory read/write.

Returns TRUE: Flash device ID is present and supported
FALSE: - Flash device ID is not valid..

Example

wFlashErase()

Erase one of the Flash device sectors.

Compatibility 386EX

Synopsis `#include "Ar386Ex.h" and "ArFlash.h"`

SINT16 wFlashErase (UINT16 iBlk);

Arguments **UINT16 iBlk - Block number of the flash device to erase**

Description Erase one of the Flash device sectors.

Returns SUCCESS - Block erased successfully
FLASHRANGE - Block number supplied is not valid
FLASHTIMEOUT - Flash device failed

Example

wFlashBurn()

Burn (program) data into one of the sectors in the flash device.

Compatibility 386EX

Synopsis `#include "Ar386Ex.h" and "ArFlash.h"`

SINT16 wFlashBurn (UINT16 iBlk, void huge *hpSrc, UINT32 clSrc);

Arguments **UINT16 iBlk - Block number of the flash device to erase**
void huge *hpSrc -> source data to burn
UINT32 clSrc - # of bytes of source data to burn

Description Burn clSrc bytes of data pointed to by hpSrc.

Returns SUCCESS - Block erased successfully
FLASHRANGE - Block number supplied is not valid

Example

wFlashRead()

Read data from the Flash device

Compatibility 386EX

Synopsis `#include "Ar386Ex.h" and "ArFlash.h"`

SINT16 wFlashRead (UINT16 iBlk, void huge *hpTrg, UINT32 clTrg);

Arguments **UINT16 iBlk - Block number of the flash device to read**
void huge *hpTrg -> buffer to receive flash data
UINT32 clTrg - # of bytes to read

Description Read data from the Flash device

Returns SUCCESS - Block erased successfully
FLASHRANGE - Block number supplied is not valid
FLASHTIMEOUT - Flash device failed

Example

fFlashChk()

Check the part ID of the Flash part installed

Compatibility 386EX

Synopsis

```
#include "Ar386Ex.h" and "ArFlash.h"

FLAG  fFlashChk  (void);
```

Arguments None

Description Check the part ID of the Flash part installed

Returns TRUE: Flash device ID is present and supported.
FALSE: Flash device ID is not present or supported.

Example

uFlashId()

Return the ID of the installed flash device.

Compatibility 386EX

Synopsis `#include "Ar386Ex.h" and "ArFlash.h"`

UINT16 uFlashId (void);

Arguments **None**

Description Check the part ID of the Flash part installed

Returns The flash ID

Example

Serial Communications Library

Introduction

Serial communications are one of the most important functions provided by a development toolkit for embedded systems. Often serial communications interface are the only link between an embedded system and the outside world. The serial communications drivers provided by these functions provide hard real time interrupt driven serial communications. The drivers support from one to sixty four simultaneous serial communications ports with a multitude of system utilities provided to ease the interface and implementation of even the most demanding serial interface project.

Special consideration has been given to the functionality offered by the functions in the library with regards to use with embedded systems. Since many embedded serial communications protocols are half duplex in nature, the interface library offers block transmission of data such that entire buffers of data can be moved into the communications queues instead of the overhead of sending one character at a time.

The low level drivers are optimized for use with Arcom's line of serial interface boards. All drivers are fully interrupt driven and work in conjunction with the RTK.

Regardless of the target system that one chooses to use, the interface to the serial communications functions are homogeneous across the entire product platform. Code written for a '188 based target processor can be moved across platforms to a '386 or '486 based hardware platform without ANY code modifications.

Macros and Constants

The following are the Macros and constants defined for the Serial Communications libraries.

Synopsis #include "ArComm.h"

Return Codes:

COMMNOTOPEN	(-1)	Port not opened for access
COMMRANGE	(-2)	Parameter supplied is out of range.
COMMNOTINIT	(-4)	wArInitCpuComs() was not called or was unsuccessful prior to operation.
COMMNORAM	(-5)	Insufficient RAM exists for operation.
COMMOVERFLOW	(-6)	Formatted string > 255 characters
COMMBFRFULL	(-7)	Async buffer is full
COMMTBLFULL	(-8)	Maximum number of off-board serial ports have been initialised.
COMMBFREMPTY	(-9)	No characters are in the async. buffer

Parity Parameters

PARITY_NONE	-	Ignore parity
PARITY_EVEN	-	Count of on bits must be even
PARITY_ODD	-	Count of on bits must be odd
PARITY_SPACE	-	Parity bit is always off (0)
PARITY_MARK	-	Parity bit is always on (1)

Handshake Parameters See function wArOpen()

Example See example program demo[386|188].c

See Also

wArInitCpuComs()

Initialises the onboard serial communications ports.

Synopsis

```
#include "ArComm.h"

SINT16 wArInitCpuComs( void );
```

Arguments none

Function Initialises the onboard serial communications drivers for use within Arcom's serial communications library. This function must be called before any subsequent communications functions may be accessed. This can only be called once.

Returns The number of serial ports found.

Example See example program *demo[386|188].c*

See Also

wArOpen()

Open a serial communications port.

Synopsis

```
#include "ArComm.h"

SINT16 wArOpen    (UINT16 uPort,
                   UINT32 ulBaud,
                   UINT8  bParity,
                   UINT8  bWordlen,
                   UINT8  bStopBit,
                   UINT16 cRxBuf,
                   UINT16 cTxBuf,
                   SINT16 wWarmUp,
                   SINT16 wWarmDown
                   );
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

UINT32 ulBaud
Baud rate. Supported values include 300L through 115200L. Note that some of the controller hardware will not support all baud rates. Refer to the appropriate hardware controller manual.

UINT8 bParity
Parity configuration. Valid parameters are PARITY_NONE, PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE

UINT8 bWordlen
Port word length. Valid parameters are 5 - 8

UINT8 bStopBit
Port stop bits. Valid parameters are 1 or 2.

UINT16 cRxBuf
This value define the size of the receive buffer. Buffers are dynamically located during this call. Valid parameters are 2 .... 32767. Subsequent calls for the same port will re-allocate this buffer if it is a different size than the original.

UINT16 cTxBuf
This value define the size of the receive buffer. Buffers are dynamically located during this call. Valid parameters are 2 .... 32767. Subsequent calls for the same port will re-allocate this buffer if it is a different size than the original.

SINT16 wWarmUp
SINT16 wWarmDn

Defines hardware handshaking mode.
Warm up/down delays in ms.
(Only available on Target188,
Target 386 implements only manual handshaking.)

If wWarmUp is set to equal -1 the RTS line is not set upon Tx.
If both wWarmUp and wWarmDn are set to equal zero, automatic handshaking is enabled (Target188).

Note that when automatic handshaking is enabled the following characteristics will apply to the communications channel.

- **+ RTS will automatically be asserted whenever any characters are in the transmit buffer. Once the transmit buffer is completely empty and one bit time after the last byte has been shifted out of the SCC, RTS will be deasserted.**
- **+ CTS will control the transmitter. No characters will be shifted out of the transmit buffer until the CTS lead is true.**
- **+ DCD will control the receiver in that no characters will be shifted into the receive buffer until the DCD lead is true.**

Only ports that use SCC8530 UARTS, Target188, will use the AUTO type of serial interface handshaking. 16450 device ports will use the MAN handshaking regardless of the passed argument.

Function Opens and initialises serial communications port. wArOpen() can be called multiple times for the same channel to set different parameters of operation such as buffer size, baud rate, parity, etc. Any pending communications that were in progress on a channel at the time wArOpen() is called will be terminated.

Returns

SUCCESS	(0)	Serial communication port opened
COMMRANGEERR	(-2)	Port specified is beyond initialised ports
COMMNOTINIT	(-4)	Port specified is not initialised
COMMNORAM	(-5)	Not enough memory available for buffers.

Example See example program *demo[386|188].c*

See Also

wArPrintf()

Print formatted string out serial communications port.

Synopsis

```
#include "ArComm.h"

SINT16 wArPrintf ( UINT16 uPort, UCHAR *sFormat, ...);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

UCHAR *sFormat
Format specifier string. Refer to the Borland C printf() function documentation for specifier string descriptions.

ellipsis (...)
Optional arguments as specified within format string.

Function

wArPrintf() accepts a serial communications port number, and a series of arguments and outputs the formatted data as specified by the format specifier string given by the format.

There must be the same number of format specifiers as arguments. wArPrintf() uses the Borland C sprintf() function to format the data before sending the resulting string to the serial communications port buffer. Refer to the Borland manual for the printf() format strings that may be used.
The **maximum** number of characters that can be printed at one time is **255**. If more that 255 characters need to be printed, multiple calls to wArPrintf() must be used.

Returns

Positive number	(>= 0)	Characters buffered for output
COMMNOTOPEN	(-1)	Port not opened for access
COMMOVERFLOW	(-6)	Formatted string > 255 characters

Warning

COMMOVERFLOW indicates the static format buffer has been overflowed. The results are unpredictable but may cause system wide instability.

Example

See example program *demo[386|188].c*

See Also

wArPutc(), wArBlkTx

wArPutc()

Output a character to a serial communications port.

Synopsis

```
#include "ArComm.h"

SINT16 wArPutc (UINT16 uPort, SINT16 wChr);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

SINT16 wChr
Character to send to specified communications port.

Function

wArPutc() takes the passed character and places it into the transmit buffer. If the buffer was empty prior to this call, the transmitter is re-enabled along with the transmit interrupts. The character is taken from the buffer and placed into the UART as soon as a buffer in the UART becomes available. If the automatic handshaking mode was chosen for this channel in wArOpen(), the character will be buffered until such a time as the CTS lead becomes true.

Returns

SUCCESS	(0)	Character was successfully placed into the buffer
COMMNOTOPEN	(-1)	Port not opened for access
COMMBFRFULL	(-7)	Insufficient space in transmit buffer for character.

Example

See example program *demo[386|188].c*

See Also

wArGetc()

Get a character from the specified serial communications port.

Synopsis

```
#include "ArComm.h"

SINT16 wArGetc ( UINT16 uPort );
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArGetc() gets the next available character from the receive buffer.

Returns

Zero to 255 (0 - 0xff)	>= 0	Character read from receive buffer
COMMNOTOPEN	(-1)	Port not opened for access
COMMBFREMPTY	(-9)	No characters are in the receive buffer

Example

See example program *demo[386|188].c*

See Also

wArAllSent()

Determine if the transmit buffer and transmitter shift register are empty.

Synopsis

```
#include "ArComm.h"

SINT16 wArAllSent ( UINT16 uPort);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArAllSent() checks the ALL SENT status of the passed port. This bit is an indications that ALL bytes have been emptied from the UARTS FIFO buffers as well as all data shifted out of the transmit shift registers. For application programs where the RTS/CTS handshaking signals are manipulated from within the application, this function can be used to check for confirmation that all data has been shifted out of the UART.

Returns

FALSE	(0)	**Not all** characters have been sent
Greater than 0	(> 0)	**All** character have left the UART
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArBlkTx()

Put a block of characters to transmit buffer.

Synopsis

```
#include "ArComm.h"

SINT16 wArBlkTx( UINT16 uPort,
                 UCHAR *pBfr,
                 UINT16 cBfr );
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

UCHAR *pBfr
Pointer to characters to send any value 0 to 0ffh is valid.

UINT16 cBfr
Number of bytes to place into the transmit buffer.

Function

wArBlkTx() places "cBfr" count of characters from pBfr into the transmit buffer. If TOO many characters are tried to be placed into the transmit buffer NOTHING is entered and COMMBUFFERFULL status is returned. If the automatic handshaking mode was chosen for this channel in ArOpen(), the character will be buffered until such a time as the CTS lead becomes true.

Returns

SUCCESS	(0)	Characters were successfully placed into the buffer
COMMNOTOPEN	(-1)	Port not opened for access
COMMBFRFULL	(-7)	Insufficient space in transmit buffer for all characters.

Example

See example program *demo[386|188].c*

See Also

wArPutc()

wArBlkRx()

Retrieve specified number of characters from receive buffer

Synopsis

```
#include "ArComm.h"

SINT16 wArBlkRx (  UINT16 uPort,
                   UCHAR *pBfr,
                   UINT16 cIn);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports
UCHAR *pBfr
Pointer to characters to send any value 0 to 0ffh is valid.
UINT16 cIn
Number of bytes to received.

Function wArBlkRx() Retrieves exactly "cIn" characters from receive buffer.
If insufficient characters are present, NONE are retrieved

Returns

Greater than or equal to zero	>= 0	Number of characters retrieved equal to zero
COMMNOTOPEN	(-1)	Port not opened for access

Example See example program *demo[386|188].c*

See Also wArGetc()

wArDcd()

return the state of the Data-Carrier-Detect (DCD) handshaking lead.

Synopsis

```
#include "ArComm.h"

SINT16 wArDcd (UINT16 uPort)
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArDcd() returns the current state of the DCD handshaking lead of the passed channel.

Returns

FALSE	(0)	DCD is not asserted
Greater than zero	(> 0)	DCD is asserted
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArDtr()

Assert/Deassert Data-Terminal-Ready (DTR) handshaking lead.

Synopsis

```
#include "ArComm.h"

SINT16 wArDtr (UINT16 uPort, FLAG fState)
```

Argument

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports
FLAG fState
Desired DTR state. 0 == Deassert !0 == Assert

Function

wArDtr() forces the DTR handshaking lead to be asserted or deasserted depending upon the passed value of "wState".

Returns

SUCCESS	(0)	Call was successful.
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArRts()

Assert/Deassert Request-To-Send (RTS) handshaking lead.

Synopsis

```
#include "ArComm.h"

SINT16        wArRts(UINT16 uPort, FLAG fState);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports
FLAG fState
Desired DTR state. 0 == Deassert !0 == Assert

Function wArRts() forces the RTS handshaking lead to be asserted or deasserted depending upon the passed value of "state". Note that when the channel has been initialised in the automatic handshaking mode, wArCpuRts() can be used to assert RTS before any characters are placed into the transmit buffer but cannot be used to properly deassert RTS. This aspect of the function can be used to obtain a longer warm-up or key time on certain modems and radios that the automatic handshaking will.

Returns

SUCCESS	(0)	Call was successful.
COMMNOTOPEN	(-1)	Port not opened for access

Example See example program *demo[386|188].c*

See Also

wArRxBst()

Get number of characters in the receive buffer.

Synopsis

```
#include "ArComm.h"

SINT16        wArRxBst (UINT16 uPort);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArRxBst() returns the number of characters in the receive buffer.

Returns

Greater than zero	(> 0)	Receive buffer free byte count
Zero	(0)	Receive buffer is empty
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArTxBst()

Get number of bytes free in transmit buffer.

Synopsis

```
#include "ArComm.h"

SINT16        wArTxBst(UINT16 uPort);
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArTxBst() returns the number of bytes free in the transmit buffer.

Returns

Greater than zero	(> 0)	Transmit buffer free byte count
Zero	(0)	Transmit buffer is full
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program demo[386|188].c

See Also

wArAllSent()

wArCts()

Get current state of Clear -To-Send (CTS) handshaking lead.

Synopsis

```
#include "ArComm.h"

SINT16 wArCts(UINT16 uPort)
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArCts() returns the current state of the CTS handshaking lead of the passed port

Returns

0 if CTS of passed channel is false else positive if true.
COMMNOTOPEN (-1) - Channel not open for access.

Returns

FALSE	(0)	CTS lead is not asserted.
TRUE	(1)	CTS lead is asserted.
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArRxFlush()

Flush current contents of the receive buffer.

Synopsis

```
#include "ArComm.h"

SINT16 wArRxFlush(UINT16 uPort)
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArRxFlush () flushes the current contents of the receive buffer.

Returns

SUCCESS	(0)	Call was successful.
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also

wArTxFlush()

Flush current contents of the receive buffer.

Synopsis

```
#include "ArComm.h"

SINT16 wArTxFlush(UINT16 uPort)
```

Arguments

UINT16 uPort
Communications channel number: 0 - Maximum Initialised physical ports

Function

wArTxFlush() flushes the current contents of the transmit buffer. If any communications were pending, the characters currently in the UART buffer will be transmitted. Auto handshaking if enabled will still function correctly.

Returns

SUCCESS	(0)	Call was successful.
COMMNOTOPEN	(-1)	Port not opened for access

Example

See example program *demo[386|188].c*

See Also